Design Optimization of Computing Systems

Assignment 1 - Benchmarking the Lua GC

Deadline: 11:59 PM, 26th August, 2024 **Maximum Marks : 100**

Lua is a lightweight, high-level programming language designed for embedding within applications. It is known for its simplicity, efficiency, and small footprint, making it ideal for use in embedded systems, game development, and scripting. Lua features a clean syntax, dynamic typing, and automatic memory management through garbage collection. It's highly extensible, allowing integration with C/C++ libraries, and supports cross-platform development. Lua's versatility and ease of embedding have made it a popular choice for adding scripting capabilities to a wide range of software applications. Another fact, Lua's interpreter is written in **C** language.

In this assignment you will be playing around with **Lua's Garbage Collector**. More specifically, you will be understanding how Lua's GC works, followed by benchmarking it. To give a very brief background, Lua's GC supports the traditional *stop the world* garbage collection as well as modern GC techniques like **generational** and **incremental** GC, the latter being the default configuration. This assignment is supposed to give you an idea regarding:

- how to read and understand the source code of any program.
- how to benchmark any program and identify the potential problem areas.

The assignment is presented in 2 parts as follows:

Part 1 - Understanding Lua's GC

In this part, you will be understanding how Lua's GC works by reading its source code. To begin with download and build the latest version of Lua(*i.e.* version 5.4.7) from here following the instructions. After you have built it, proceed as follows:

- You are given a sample testbench (see Appendix). You will be using this testbench for understanding and benchmarking Lua's GC.
- Write 3 C files named `fullgc.c`, `incrementalgc.c` and `generationalgc.c` which will load the testbench file and run it under different GC configurations *i.e.* full GC, incremental GC and generational GC.
  Note: Lua being written in C language allows you to load and run lua files from C. Also you are not allowed to use `fork` and `exec` system calls.
- Run `callgrind(valgrind)` on the executables, followed by parsing the output using tools like `gprof2dot` or `kcachegrind` to generate call graphs.
- Use the call graphs to read the source code and understand how Lua's GC works in different configurations.

Deliverables: Submit the three C files along with a report on how Lua's GC works in not more than 5 pages.

Part 2 - Benchmarking Lua's GC

After understanding how Lua's GC works its time to benchmark it. To do this, you will be proceeding as follows:

1. Use the `callgrind` outputs or call graphs to find the percentage of instructions consumed by GC for the different GC configurations. Report the results and explain the variation in the results.

2. Re-run the previous test under the following set of different parameters (on the same testbench):

   - $m = 100, n = 100$
   - $m = 500, n = 100$
   - $m = 5000, n = 100$

   Report the results and explain any trend that you see.

3. Run `perf` analysis on the same testbench with original parameters and report the following **metrics** for the different GC configurations and a configuration without GC:

   - Branch Misses
   - Page Faults
   - Cache Misses
   - Instructions Per Cycle

   Now answer the following questions:

   - For each of these metrics, why do you think it is relevant/not relevant for this scenario?
   - Explain the variation in the values of these metrics across different configurations of GC and the configuration with no GC.

Deliverables: Submit a single report which contains all the results, explanations and answers as asked in part 2. Also submit the call graph images and `callgrind` outputs for each of the parameter sets(and the original parameters).

Submission Guideline: Zip all the files into a single zip file named Asgn1_<Roll Number>.zip and upload it on moodle.

Evaluation Guidelines

The total marks for this assignment = 100.
The marks breakup is given below:

Parts	Items	Marks
Part 1: Understanding Lua's GC	C files	$5 \times 3 = 15$
	Report on each GC configurations	$15 \times 3 = 45$
Part 2: Benchmarking Lua's GC	Sub Part - 1	10
	Sub Part - 2	10
	Sub Part - 3	20
Total		**100**

Appendix

Test Bench Code: Following is the testbench code. It is commented well for you to understand what it is doing.

```
-- This is a testbench. This file is to be used to benchmark Lua's GC.
-- It is a simple matrix creation of size m x n.
function test()
    local matrix = {}

    -- Parameters m and n.
    -- Change these values to test the performance of the GC.
    local m = 1000
    local n = 100

    -- Do not change the following code
    for i=1,m do
        local var = {}
        matrix[i] = var
        for j=1,n do
            local temp_var = {}
            temp_var[1] = {"123"}
            temp_var[10] = {"hello"}
            local temp_var2 = {}
            temp_var2[1] = {"178"}
            temp_var2[10] = {"world"}
            matrix[i][j] = temp_var
        end
    end
end
```